GLUT Tutorial

John Tsiombikas (Nuclear / The Lab Demos)

May 6, 2005

1 Introduction to GLUT (OpenGL Utility Toolkit)

This is an introductory text, for those who are interested in using the OpenGL library in a cross-platform manner. OpenGL by itself is not tied to any specific platform/OS, however there are some things that must be done in order to work with OpenGL, such as initialization of the underlying window system, for example Connection to an X server, and event handling, that are inherently system-specific. In order to overcome that difficulty, there are various libraries that provide a common interface to that kind of operations, hiding the system-specific details from us. GLUT is maybe the most well-known and widely used such library.

Currently there are two implementations of GLUT as far as I know, the original glut by Mark Kilgard for X *(ported to win32 by Nate Robins and Paul Mayfield)*, and the new freeglut project which is a replica of glut under a free software license *(the X-Consortium license)* which, in contrast to the original glut license, does allow modification and redistribution of the library.

2 Obtaining GLUT

The original GLUT library can be obtained from the opengl.org website, from `http://www.opengl.org/resources/libraries/glut/glut_downloads.html` in source code packages for UNIX/X11 and Windows.

Binary precompiled versions are available for SGI, Solaris, and Windows, links to those are available at the same page as the source distribution I mentioned.

3 Obtaining FreeGLUT

FreeGLUT can be obtained in source code form for UNIX and Windows at the freeglut project website: `http://freeglut.sourceforge.net/`

Precompiled binaries available in most GNU/Linux distributions, for example on debian "`apt-get install freeglut3-dev`", suffices to install

it, also on windows there is a easily installable package for Bloodshed's DevC++ (link available on the freeglut website under "downloads").

4 Preparing to use GLUT/FreeGLUT

From now on, I will refer to both these libraries as just glut.

To use glut with gcc under GNU/Linux, and probably other UNIX variants, you just have to include `<GL/glut.h>` at your program and pass the `-lglut` option during linking. That implies that glut is installed at the default system directories (usually /usr/include/GL for the header files, and /usr/lib for the shared library), or anywhere on the compiler/linker search paths for that matter, otherwise you have to specify the directories using `-I` and `-L` flags.

To use glut under MS Windows with MS Visual C++ you have to create a Win32 console application project and add glut32.lib to your project using the "Add→existing item" of the "solution explorer" (every reference to exact GUI actions in this document are for Visual Studio.NET).

Now the include part gets more complicated as we move into the windows domain and the programming environment gets more braindamaged, you could add the path to the glut headers at the compiler's search paths by doing something like: "Tools→Options: Projects→VC++ Directories (Show directories for Include files)" and then including `"glut.h"` but that is a fundamendally wrong way of doing it since these options are not preserved when giving the project files to someone else or transfering to another computer.

Another much better way is by going to "Project→properties", while having the current project selected on the solution explorer, and then going to "C/C++→Additional Include Directories" and adding the path to the glut headers. This is better since it is saved in the project files and thus preserved when moving the project, but both ways are wrong in respect to cross-platform development.

The most sane approach I can think of, is putting the actual glut headers inside the compiler's standard include directories in the subdirectory GL along with gl.h and glu.h that should be there, which enables you to just include `<GL/glut.h>` as is customary and get over it. This of course requires again that you must do that and/or provide directions for everyone involved in the project to do the same thing on their machines as well.

Bottom-line, everything would be much easier if people would just stick to UNIX.

5 Using GLUT

Glut is very simple to use, the main idea is that you initialize it, create a window, setup callback functions for various events and then let it handle everything. But let's see an example that will clear things up. Check the comments in the sample code for details.

```
#include <stdlib.h>
#include <GL/glut.h>

void update_display(void);
void key_handler(unsigned char key, int x, int y);

int main(int argc, char **argv) {

    /* glutInit() initializes the library and handles window-
     * system specific initialization. It may intercept some
     * command line args that glut recognizes and remove them
     * from the array.
     */
    glutInit(&argc, argv);

    /* set the size of the window that will be created by
     * the call to glutCreateWindow() further on.
     */
    glutInitWindowSize(800, 600);

    /* specify display mode, some usefull flags are:
     * GLUT_DOUBLE: use double buffering
     * GLUT_DEPTH: provide a depth buffer (Z-buffer)
     * GLUT_STENCIL: provide a stencil buffer
     */
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_STENCIL);

    /* create the actual window with the specified title */
    glutCreateWindow("GLUT test");

    /* specify a function to be called when glut needs to
     * redraw the window.
     */
    glutDisplayFunc(update_display);

    /* specify a function to handle keyboard events */
    glutKeyboardFunc(key_handler);
```

```
    /* specify a function to be called when idle
     * (i.e. no other events are to be handled)
     * */
    glutIdleFunc(update_display);

    /* do some usual OpenGL setup */
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CW);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glMatrixMode(GL_PROJECTION);
    gluPerspective(45.0, 1.3333, 1.0, 1000.0);

    /* enter glut main event loop, this will call the
     * callbacks we provided on various events and never
     * return, it will exit when the window is closed by
     * calling exit(), so if we want to perform any
     * cleanup before the program exits, we can use
     * atexit() to register a cleanup function.
     */
    glutMainLoop();
    return 0;
}

/* This is the function we gave to glut as display and
 * idle callback it will be called whenever glut needs to
 * update the display and when it has nothing else to do.
 * Usual OpenGL frame drawing should go in here.
 */
void update_display(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClearDepth(1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -4);
    glRotatef(35.0, 1.0, 1.0, 0.0);

    /* glutSolidTeapot() draws a typical Utah Teapot */
```

```
    glutSolidTeapot(1.0);

    /* glutSwapBuffers() shows the back buffer */
    glutSwapBuffers();
}

/* this is our keyboard callback, first argument is the
 * key that was pressed, next two arguments are the current
 * mouse position
 */
void key_handler(unsigned char key, int x, int y) {
    if(key == 27) {
        exit(0);
    }
}
```

The code should be easy to understand, after all it's being commented to death. Glut provides a lot of different handlers from timing, to mouse events and specialized input devices, and even a popup menu management system.

This example program displays the graphics in a window, if you want to make your program fullscreen the only addition that you must make is call the glutFullScreen() function somewhere.

6 What about OpenGL extensions

Glut provides a very handy function that checks to see if a specified OpenGL extension is supported, just call glutExtensionSupported() passing a string with the name of the extension you want as argument to the function.

Example:

```
if(glutExtensionSupported("GL_ARB_transpose_matrix")) {
    /* do whatever you need to do */
}
```

Unfortunately the original glut by Mark Kilgard does not provide a function to retrieve pointers to functions of OpenGL extensions so one has to go through the usual, platform-specific way of getting them (more details below).

Luckily the new freeglut library provides such a mechanism, we just have to call the glutGetProcAddress() function, passing as argument a string with the name of the function we want (note: in order to use that function you have to include `<GL/freeglut.h>` rather than `<GL/glut.h>`).

Example:

```
PFNGLLOADTRANSPOSEMATRIXFARBPROC glLoadTransposeMatrixfARB;
glLoadTransposeMatrixfARB = (PFNGLLOADTRANSPOSEMATRIXFARBPROC)
         glutGetProcAddress("glLoadTransposeMatrixfARB");
```

Now if you are using the original glut that does not provide such a function for retrieving extension functions in a platform independent way, you must use the appropriate platform-specific functions, so the best thing is to do the following preprocessor trick:

```
#ifdef __unix__

#include <GL/glx.h>
#include <GL/glxext.h>
#define GetProcAddress    glXGetProcAddress

#else /* assume windows */

#include <windows.h>
#define GetProcAddress    wglGetProcAddress

#endif /* __unix__ */
```

That's about it, for additional information check out the documentation at opengl.org and freeglut website.